Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Gantar

Razvoj postopka za nadzor kakovosti grafitnih polizdelkov na osnovi njihovih slik

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Danijel Skočaj
Somentor: izr. prof. dr. Bogdan Filipič

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V industrijski proizvodnji komutatorskih elektromotorjev je zelo pomemben korak natančna izdelava komutatorja oz. spajanje dveh njegovih sestavnih delov. Za uspešno delovanje je v proizvodnem procesu potrebno zagotoviti učinkovit nadzor kakovosti izdelanih komutatorjev. V diplomski nalogi razvijte vgradno aplikacijo za avtomatski nadzor kakovosti proizvodnje komutatorjev. Osredotočite se na razvoj ustreznih algoritmov strojnega vida. Postopek naj bo sposoben iz zajetih slik komutatorjev pridobiti ustrezne atribute, nato pa z metodami strojnega učenja, predvsem z uporabo odločitvenih dreves, zgraditi ustrezen klasifikator, ki bo omogočal zaznavanje napak. Razviti postopek tudi ustrezno ovrednotite in ocenite njegovo uporabnost v industrijski proizvodnji.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Gantar sem avtor diplomskega dela z naslovom:

Razvoj postopka za nadzor kakovosti grafitnih polizdelkov na osnovi njihovih slik

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Danijela Skočaja in somentorstvom izr. prof. dr. Bogdana Filipiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 12. avgusta 2015 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Danijelu Skočaju in somentorju izr. prof. dr. Bogdanu Filipiču za usmerjanje in pomoč pri izdelavi diplomskega dela. Hvala tudi podjetju Kolektor Group d.o.o. za testne podatke in informacije, ki so bile uporabljene v tem delu.

Posebna zahvala gre staršem, ki so mi omogočili študij in mi tekom le-tega stali ob strani.

Kazalo

Povzetek

Komutator je pomemben in zelo občutljiv del komutatorskega elektromotorja, ki pritrjen na os motorja skrbi za periodično menjavanje smeri električnega toka in tako omogoča delovanje motorja. Kakovost izdelave komutatorja je zato odločilnega pomena za kakovost elektromotorja. Ročni nadzor kakovosti je časovno zahteven in nezanesljiv, zato je v ključnih korakih proizvodnje komutatorjev smiselna uvedba avtomatskega nadzora kakovosti. V proizvodnji grafitnih komutatorjev, ki jih sestavljata grafitna ploščica in bakrena osnova, je ključen korak spajanje teh dveh sestavnih delov. Diplomsko delo obravnava razvoj vgradne aplikacije za avtomatski nadzor kakovosti grafitnih komutatorjev po spajanju grafitne ploščice z bakreno osnovo. Namen aplikacije je prepoznavanje štirih vrst napak, do katerih pride v postopku spajanja. Najprej z metodami strojnega vida iz slik komutatorjev pridobimo atribute, nato pa na njihovi podlagi s strojnim učenjem zgradimo odločitvena drevesa, ki omogočajo določanje napak na komutatorjih. Na koncu preizkusimo še druge metode učenja in njihove rezultate primerjamo z rezultati odločitvenih dreves.

Ključne besede: nadzor kakovosti, elektromotor, strojni vid, strojno učenje, odločitveno drevo.

Abstract

A commutator is an important and very sensitive part of the commutator electric motor. Located on the motor axis, it periodicaly changes the direction of the electric current, enabling the motor to run. For this reason, the quality of commutator production is crucial for the quality of the electric motor. Manual quality control is time-consuming and unreliable, therefore it is reasonable to introduce automated quality control in the key steps of commutator production. The graphite commutator consists of two main parts, a metalized graphite disc and a copper base. One of the crucial steps in graphite commutator production is soldering of these parts. This thesis deals with the development of an embedded application for automated inspection of the commutator quality after soldering of the metalized graphite disc and the copper base. The goal of the application is to detect four types of defects occurring during the soldering process. Methods of machine vision are used first to acquire attributes from the commutator images. From these attributes decision trees are then constructed through machine learning that make it possible to determine defects on commutators. Finally, other learning methods are tested and their results compared with the results of decision trees.

Keywords: quality control, electric motor, machine vision, machine learning, decision tree.

Poglavje 1

Uvod

Računalniški vid se v povezavi s strojnim učenjem pogosto uporablja na področjih, kot so medicina, astronomija, meteorologija, robotika in industrija. Razlog za njegovo vse večjo uporabnost je predvsem napredek v zmogljivosti temu namenjene strojne in programske opreme. V industriji se kombinacija računalniškega vida in strojnega učenja uporablja predvsem za nadzor kakovosti. Tovrstni nadzor kakovosti se čedalje bolj uveljavlja zaradi njegovih prednosti v primerjavi z ostalimi metodami. Je hiter, učinkovit ter zagotavlja točne in objektivne rezultate. Rezultati omogočajo tudi arhiviranje in nadaljnjo analizo, njihova pridobitev pa ne vpliva na potek proizvodnje. Ena redkih pomanjkljivosti te metode je odvisnost od svetlobnih in drugih zunanjih pogojev, zato za nekatere naloge nadzora kakovosti ni primerna.

Diplomsko delo je nastalo v sklopu mednarodnega projekta Cognitive and Perceptive Cameras (COPCAMS) [1], v katerega sta vključena slovenska partnerja Institut "Jožef Stefan" in Kolektor Group d.o.o. Namen projekta je zasnova in praktična uporaba sistemov strojnega vida na osnovi mnogojedrnih programabilnih platform STHORM [2]. Diplomsko delo se osredotoča na uporabo kombinacije strojnega vida in strojnega učenja za namene nadzora kakovosti komutatorjev v serijski industrijski proizvodnji. Zahteve po zanesljivosti so pri izdelavi komutatorjev zelo stroge, zato je treba kakovost izdelka nadzirati že med samo izdelavo [3]. Eden od korakov v procesu izdelave

komutatorja je spojitev metaliziranega grafita z bakreno osnovo. Kakovost končnega izdelka je neposredno odvisna od kakovosti spoja, zato je spojitev eden ključnih korakov v proizvodnji in tako najbolj smiseln za vpeljavo avtomatskega nadzora kakovosti.

Postopek nadzora kakovosti komutatorjev obsega fazo strojnega vida in fazo strojnega učenja. V prvi fazi fotografije komutatorjev obdelamo in iz njih izluščimo želene atribute, v drugi fazi pa pridobljene atribute z metodami strojnega učenja uporabimo za napoved kakovosti. Postopek nadzora kakovosti je bil predhodno zasnovan na višjem nivoju v programskih okoljih LabVIEW [4] in Weka [5], v tem diplomskem delu pa smo postopek paralelizirali ter z uporabo programske knjižnice OpenCV [6] in ogrodja OpenCL [7] prenesli na nižji nivo, torej na nivo vgradnega sistema za nadzor kakovosti, osnovanega na platformi STHORM. Razvoj aplikacije je potekal na osebnem računalniku, paralelizacija pa poteka preko grafične kartice.

Diplomsko delo poleg uvoda sestavlja še pet poglavij. Poglavje 2 opisuje problem, ki ga diplomsko delo obravnava, ter motivacijo za njegovo rešitev. Predstavljeni so komutatorji, njihova proizvodnja ter napake, ki se pojavljajo pri spajanju grafitne ploščice in bakrene osnove. Uporabljena metodologija je predstavljena v poglavju 3. Prvi del poglavja se osredotoča na metode strojnega vida, drugi del pa na metode strojnega učenja. Ker sta področji strojnega vida in strojnega učenja zelo obsežni, smo se omejili na metode, ki smo jih uporabili pri razvoju sistema za nadzor kakovosti komutatorjev. V poglavju 4 so najprej predstavljena uporabljena orodja in programske knjižnice, v nadaljevanju pa je opisana implementacija vgradnega sistema za nadzor kakovosti komutatorjev. Predstavljamo jo v dveh delih. Prvi del opisuje postopek pridobivanja atributov iz slike, drugi del pa postopek klasifikacije komutatorja v ustrezen razred na podlagi pridobljenih atributov. V poglavju 5 so najprej opisane uporabljene metode in načini učenja, nato pa so predstavljeni in ovrednoteni njihovi rezultati. V zaključnem poglavju 6 povzamemo obravnavani problem, opravljeno delo in dobljene rezultate ter nakažemo možnosti nadaljnjega dela.

Poglavje 2

Opis problema

2.1 Električni komutator

Izdelava komutatorjev je glavni program podjetja Kolektor Group d.o.o. Komutator je eden ključnih delov enosmernega elektromotorja. Pritrjen je na os motorja, kjer skrbi za periodično menjavanje smeri električnega toka in tako omogoča delovanje motorja. Komutatorji so najobčutljivejši del motorja, saj so stalno izpostavljeni mehanskim in električnim silam. Ravno zaradi ključne vloge pri delovanju motorja in težkih pogojev obratovanja je kakovost izdelave komutatorjev izjemnega pomena.

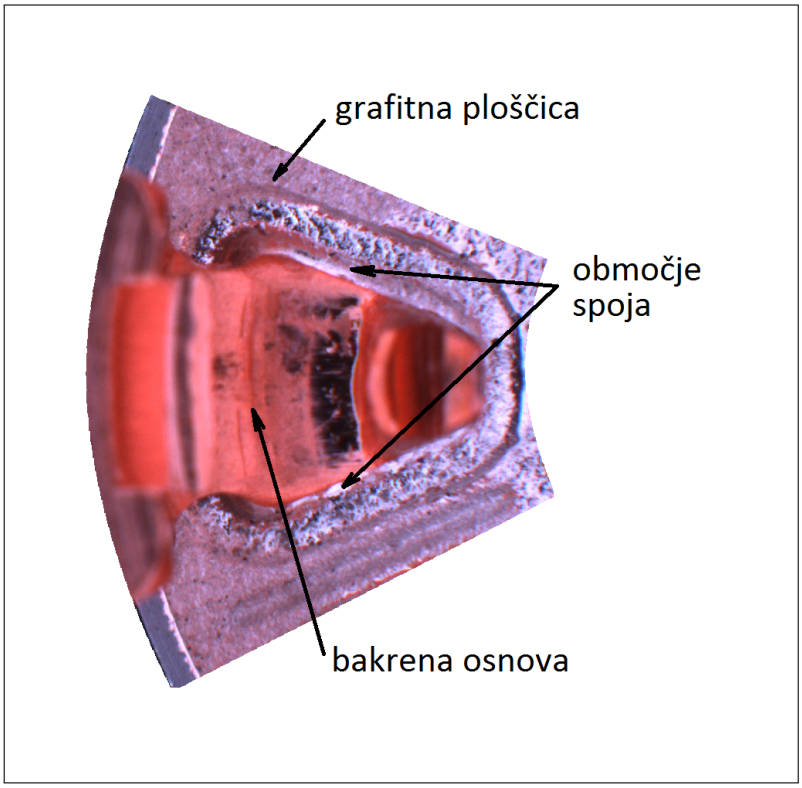
Kolektor proizvaja več vrst komutatorjev, ki se razlikujejo glede na uporabljene materiale in način izdelave. Ena izmed vrst komutatorjev je izdelana s tehnologijo spajanja grafita in bakra. Tovrstni komutatorji HPG (glej sliko 2.1) so sestavljeni iz grafitne ploščice in z njo spojene bakrene osnove. Za razliko od ostalih vrst komutatorjev, ki za drsno površino kontaktnih ščetk uporabljajo baker, pri komutatorjih HPG drsno površino predstavlja grafit, ki je v nekaterih primerih ustreznejši. Komutatorji HPG se uporabljajo v motorjih črpalk za klasična in alternativna goriva. Ker je v teh gorivih grafit obstojnejši od bakra, se življenjska doba črpalke bistveno podaljša. Poleg tega je zaradi manjšega trenja med ščetko in grafitno površino izkoristek moči boljši, zmanjšajo pa se tudi iskrenje in radiofrekvenčne motnje.

Slika 2.1: Primer komutatorja HPG, ki ga izdelujejo v Kolektorju.

Eden od ključnih korakov v proizvodnji komutatorjev HPG je spajanje metaliziranega grafita in bakrene osnove [3]. Postopek spajanja s pomočjo spojne paste združi grafitno ploščico z bakreno osnovo. Postopek mora zagotoviti čim večjo trdnost spoja, saj ta neposredno vpliva na kakovost celotnega komutatorja. Zaradi velike odvisnosti kakovosti končnega izdelka od trdnosti spoja je po končanem spajanju nujna ocena kakovosti spoja, ki trenutno poteka ročno. Ročni nadzor je časovno potraten in zaradi možnosti človeške napake ne zagotavlja točnosti rezultata, zato je smiselna avtomatizacija nadzora kakovosti.

2.2 Zajem slik

Komutator HPG je sestavljen iz osmih simetričnih segmentov. Zajem slike vsakega od segmentov je izveden s pomočjo za to razvitega manipulatorja. Manipulator vsak komutator zavrti okoli njegove osi in v ustreznih trenutkih zajame slike posameznih segmentov. Ko zajete slike, kot je opisano v razdelku 4.2.1, še nekoliko poravnamo, so pripravljene na nadaljnje procesiranje. Slika 2.2 prikazuje območje segmenta ki nas v nadaljevanju zanima. V tem območju se namreč nahajajo vsi pokazatelji morebitnih napak, ki jih iščemo. Poleg tega so na sliki označeni glavni deli segmenta, na katere se bomo sklicevali pri opisu napak v podpoglavju 2.3.

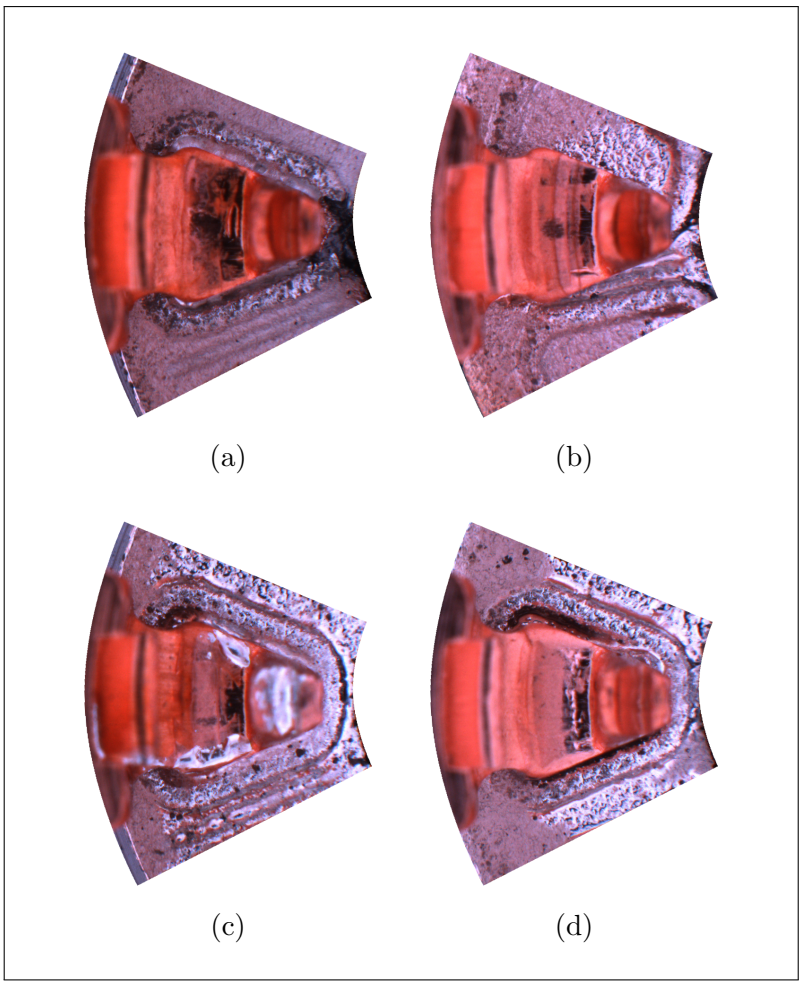


Slika 2.2: Območje segmenta, na katerem se ob morebitnih napakah nahajajo njihovi pokazatelji.

2.3 Napake v postopku spajanja

Pri spajanju grafita in bakra lahko pride do različnih vrst napak. Za namen avtomatskega preverjanja kakovosti smo se odločili za iskanje štirih najpogostejših vrst napak, ki so obenem tudi najprimernejše za zaznavanje s strojnim vidom. Komutator, ki ima vsaj eno izmed teh napak, ni primeren za nadaljnjo obdelavo in ga je potrebno izločiti s proizvodne linije. Primere iskanih napak vidimo na sliki 2.3, v nadaljevanju pa so naštete in na kratko opisane:

- poškodba metalizacije – pred ali med spajanjem pride do fizičnih poškodb in deformacij metaliziranega grafita. Na slikah se to kaže predvsem v temnejših območjih na grafitnem delu komutatorja;
- neorientiranost – grafitna ploščica in bakrena osnova po spajanju nista ustrezno poravnani;
- presežek spojne paste – pri spajanju grafitne ploščice z bakreno osnovo je bilo dodano preveč spojne paste, zato je ta na določenih mestih prekrila baker. Na slikah so glavni pokazatelji te napake bela območja na bakrenem delu;
- primanjkljaj spojne paste – pri spajanju grafitne ploščice z bakreno osnovo je bilo dodano premalo spojne paste, zato je trdnost spoja vprašljiva. Na slikah je tako območje spoja med bakrom in grafitom zaradi pomanjkanja spojne paste temnejše kot sicer.



Slika 2.3: Primeri napak pri spajanju grafitne ploščice in bakrene osnove, ki jih želimo prepoznavati: a) poškodba metalizacije, b) neorientiranost, c) presežek spojne paste in d) primanjkljaj spojne paste.

Poglavje 3

Metodologija

Postopek preverjanja kakovosti je razdeljen na fazo strojnega vida in fazo strojnega učenja. To poglavje v nadaljevanju na kratko predstavi obe področji in podrobneje opiše metode, ki so uporabljene v postopku preverjanja kakovosti, predstavljenem v poglavju 4.

3.1 Strojni vid

Področje strojnega vida se ukvarja s tehnologijami in metodami, ki se med drugim uporabljajo pri preverjanju kakovosti in analizi izdelkov ter vodenju in nadzoru industrijskih procesov. Prvi korak v teh postopkih je ustrezen zajem slike. Zajeto sliko z različnimi metodami procesiranja slik obdelamo in iz nje izluščimo informacije, na podlagi katerih nato določamo nadaljnje akcije.

Na področju strojnega vida smo se največ ukvarjali z obdelavo slike in pridobivanjem informacij iz nje. V naslednjih podpoglavjih so po knjigah [8, 9] povzete osnove procesiranja slik in metode, ki so relevantne za nadzor kakovosti komutatorjev.

3.1.1 Digitalna slika

Sliko lahko hranimo v različnih slikovnih formatih. V tabeli 3.1 so prikazani najpogostejši slikovni formati in njihove lastnosti. V našem primeru so slike shranjene v formatu TIFF, ki je vse bolj priljubljen tudi na področju strojnega vida.

Tabela 3.1: Najpogosteje uporabljeni slikovni formati in njihove lastnosti [10].

Format	Največje število barv	Največje število odtenkov sive
TIFF	16 M	256
GIF	256	256
JPEG	16 M	256
BMP	16 M	256
PNG	256 T	65536
TGA	16 M	256

Pri obdelavi slike je le-ta predstavljena kot dvodimenzionalna matrika, v kateri vrednost elementa predstavlja barvo pripadajočega slikovnega elementa. Posamezni slikovni elementi so ponavadi predstavljeni z binarnimi vrednostmi dolžine k, kar pomeni, da lahko hranijo 2^k različnih vrednosti. Najpogosteje se pri procesiranju slik srečujemo z naslednjimi vrstami slik.

- Barvne slike – Čeprav poznamo več barvnih modelov, je večina barvnih slik predstavljena z barvnim modelom RGB (angl. Red, Green, Blue). Vsak slikovni element je pri taki sliki predstavljen s tremi barvnimi kanali, od katerih vsak hrani osembitno vrednost, ki predstavlja intenziteto ene od barvnih komponent (rdeča, zelena, modra).

- Intenzitetne slike – Za razliko od barvnih slik imajo intenzitetne slike samo en barvni kanal, ki predstavlja intenziteto slike. Intenzitetno

sliko ponavadi dobimo iz barvne slike tako, da iz nje izločimo posamezen barvni kanal ali pa v en kanal združimo povprečne vrednosti vseh barvnih kanalov.

- Binarne slike – Vsak slikovni element binarne slike je predstavljen z enim bitom, kar pomeni, da lahko vsebuje le vrednosti 0 in 1. Binarna slika je podvrsta intenzitetne slike, ki pa lahko hrani le dve vrednosti. Tako sliko dobimo z upragovanjem intenzitetne slike.

3.1.2 Točkovne operacije

Točkovne operacije spreminjajo vrednosti posameznih slikovnih elementov. Pri tem je nova vrednost elementa odvisna od prejšnje vrednosti istega elementa in od preslikovalne funkcije. Preslikovalna funkcija je pri homogenih točkovnih operacijah neodvisna od položaja slikovnega elementa, pri nehomogenih točkovnih operacijah pa se pri izračunu vrednosti upošteva tudi koordinate slikovnega elementa. Pri točkovnih operacijah moramo upoštevati, da lahko nove vrednosti elementov segajo izven intervala dopustnih vrednosti, zato jih moramo omejiti. V nadaljevanju predstavljamo najpogosteje uporabljene točkovne operacije.

- Prilagajanje kontrasta – Kontrast je razlika med intenzitetami posameznih slikovnih elementov na sliki in omogoča razločevanje objektov na sliki. Kontrast slike prilagajamo tako, da vsak slikovni element množimo z določeno konstanto. Če je konstanta večja od 1, se kontrast slike poveča, saj se razlike med vrednostmi posameznih elementov povečajo, če pa je konstanta manjša od 1, se kontrast slike zmanjša.

- Prilagajanje svetlosti – Svetlost slike pove, kolikšna je povprečna intenziteta slike. Spreminjanje svetlosti ne vpliva na razlike med slikovnimi elementi. Svetlost slike prilagajamo tako, da vsem slikovnim elementom prištejemo konstanto. Če je konstanta večja od 0, se svetlost slike poveča, če pa je manjša od 0, se zmanjša.

- Invertiranje slike – Pri invertiranju slik gre za spremembo vrstnega reda vrednosti slikovnih elementov. Sliko invertiramo tako, da stare vrednosti negiramo in rezultatom prištejemo konstanto, ki je enaka največji vrednosti, ki jo lahko hrani posamezen slikovni element.

- Upragovanje slike – Upragovanje intenzitetno sliko pretvori v binarno. Vrednosti slikovnih elementov loči na dva razreda glede na mejno vrednost. Slikovni elementi, ki so manjši od mejne vrednosti, dobijo vrednost 0, elementi, ki so večji ali enaki mejni vrednosti, pa dobijo vrednost 1.

3.1.3 Filtriranje slik

Filtriranje je operacija, ki za izračun vrednosti novega elementa uporabi več kot en slikovni element. Natančneje za izračun novega elementa uporabi slikovne elemente okoli njega. Vpliv sosednjih slikovnih elementov je določen z utežmi v jedru filtra. Učinek filtra je odvisen od njegove vrste in nastavitev. Filtre glede na njihove matematične lastnosti delimo na linearne in nelinearne.

Značilna predstavnika linearnih filtrov sta:

- filter za glajenje – jedro filtra matrika vsebuje samo pozitivne uteži, zato tak filter zmanjšuje razlike med sosednimi slikovnimi elementi in tako gladi sliko;

- diferenčni filter – jedro filtra vsebuje tudi negativne uteži, zato tak filter poudarja lokalne spremembe v intenziteti slike.

Med nelinearnimi filtri se najpogosteje uporabljajo naslednji:

- minimalni in maksimalni filter – element prevzame vrednost najmanjšega oz. največjega slikovnega elementa v območju jedra filtra;

- medianin filter – ta filter nadomesti vrednost slikovnega elementa z vrednostjo mediane elementov z območja jedra filtra.

Na rezultat filtriranja lahko vplivamo tudi s spreminjanjem filtrirnih nastavitev. Nastavljamo lahko velikost in obliko jedra filtra, hkrati pa lahko jedru določimo uteži, ki določajo, kakšen vpliv ima posamezen slikovni element regije.

3.1.4 Morfološke operacije

Morfološke operacije so namenjene predvsem binarnim slikam. Na binarnih slikah slikovni elementi z vrednostjo 0 predstavljajo ozadje, elementi z vrednostjo 1 pa ospredje slike. Morfološke operacije uporabljamo za nadzorovano spreminjanje lokalne strukture takih slik. Delovanje morfoloških operacij je odvisno od uporabljenega strukturnega elementa, ki je podobno kot slika predstavljen z binarno matriko.

Osnovni morfološki operaciji sta erozija in širitev. Z erozijo območja ospredja skrčimo in tako odstranimo manjša območja, medtem ko s širitvijo območja ospredja razširimo. Ti dve osnovni operaciji vključujeta sestavljena operatorja:

- odprtje – najprej izvedemo operacijo erozije, ki odstrani manjša območja, nato pa na dobljeni sliki izvedemo še operacijo širitve, ki neodstranjena območja razširi na prvotno velikost;
- zaprtje – najprej izvedemo operacijo širitve, ki območja ospredja razširi in zapolni morebitne luknje v njih, nato pa na dobljeni sliki izvedemo še operacijo erozije, ki povečana območja skrči na začetno velikost.

3.1.5 Označevanje območij na sliki

Pri analizi območij na binarni sliki je potrebno ugotoviti, kateremu območju pripadajo posamezni slikovni elementi ospredja slike in koliko območij je na sliki. Postopek iskanja teh podatkov se imenuje označevanje območij (angl. region labeling). Postopek označevanja območij sosednje slikovne elemente ospredja postopno poveže v območja, pri čemer vsi slikovni elementi enega območja dobijo skupno oznako.

Pri označevanju območij se lahko odločimo za dve soseščini, ki določata, kateri slikovni elementi so sosedje opazovanega slikovnega elementa. 4-soseščina za sosede upošteva samo slikovne elemente, ki se nahajajo pravokotno od opazovanega, medtem ko 8-soseščina upošteva tudi slikovne elemente, ki se nahajajo diagonalno od opazovanega slikovnega elementa.

Največkrat uporabljena postopka za označevanje območij sta poplavljanje in zaporedno označevanje, ki delujeta zaporedno, vendar se z razvojem tehnologij za vzporedno procesiranje uveljavljajo tudi vzporedni postopki označevanja območij. Pred in med postopki označevanja so oznake slikovnih elementov $U_{(u,v)}$ naslednje:

$$U_{(u,v)} = \begin{cases} 0 & \text{slikovni element ozadja,} \\ 1 & \text{neoznačen slikovni element ospredja,} \\ 2, 3, ... & \text{označen slikovni element ospredja,} \end{cases} \tag{3.1}$$

po končanem postopku pa so vsi elementi ospredja označeni z oznako območja, ki mu pripadajo. Število območij dobimo tako, da preštejemo število različnih oznak večjih od 1, velikost posameznega območja pa tako, da preštejemo slikovne elemente, ki nosijo oznako tega območja.

3.1.6 Iskanje ujemanja predloge s sliko

Na področju strojnega vida je pogost problem iskanje določenega motiva na sliki. Reševanja problema se lotimo tako, da predlogo, na kateri se nahaja iskani motiv, primerjamo z deli obravnavane slike. Predlogo pomikamo po sliki in za vsak pomik izračunamo ujemanje, ki ga predloga dosega na tem delu slike. Uporabimo lahko različne mere ujemanja, ki primerjajo intenziteto soležnih slikovnih elementov predloge in dela slike. Na koncu dobimo matriko, ki za vsak del slike pove stopnjo ujemanja s predlogo. Dobljeno matriko glede na uporabljeno mero ujemanja ustrezno upragujemo in tako najdemo mesta, na katerih je ujemanje največje in je tako verjetnost, da se tam nahaja iskani motiv, največja.

3.2 Strojno učenje

V splošnem kot strojno učenje opredelimo tiste aktivnosti, ki pripomorejo, da nek sistem bolje opravlja svoje naloge. V nasprotju s statistično analizo podatkov strojnega učenja ne zanima le gradnja čimbolj zanesljivih modelov, temveč tudi gradnja razumljivih modelov, ki jih je mogoče pojasniti. Glavni nalogi strojnega učenja sta razvoj in uporaba algoritmov za pridobivanje znanja iz podatkov. Najbolj raziskano področje strojnega učenja je učenje na osnovi primerov (angl. instance based learning), imenovano tudi induktivno strojno učenje [11]. Pri tovrstnem učenju algoritmu posredujemo učne primere, ki predstavljajo primere rešenih problemov nekega področja. Algoritem na učnih primerih išče vzorce in zakonitosti, ki povezujejo lastnosti primerov z njihovimi razredi. Ugotovitve nato uporabi za nadaljnje reševanje problemov na tem področju.

V nadaljevanju so po knjigi [12] povzete metode induktivnega strojnega učenja, ki so bile uporabljene v diplomskem delu.

3.2.1 Odločitvena drevesa

Ena najbolj znanih in široko uporabljenih metod strojnega učenja na osnovi učnih primerov je gradnja odločitvenih dreves. Rezultat učenja teh metod je predstavljen v obliki odločitvenega drevesa. Odločitveno drevo je v formalnem smislu graf, sestavljen iz vozlišč in povezav. Notranja vozlišča v grafu predstavljajo določen atribut, povezave med vozlišči predstavljajo vrednosti tega atributa, zunanja vozlišča pa vrednosti razredov.

Osnovni algoritem za gradnjo odločitvenega drevesa je algoritem ID3 [13]. Ta algoritem odločitvena drevesa gradi tako, da učno množico deli na podmnožice. Delitev poteka rekurzivno in se nadaljuje, dokler vsi primeri posamezne podmnožice ne pripadajo istemu razredu. Pri delitvi na podmnožice je ključna ustrezna izbira atributa, na podlagi katerega se opravi delitev. Izbrani atribut mora v deljeni množici nositi največjo količino informacij o tej množici, saj tako omogoča najboljšo razdelitev množice. Poznamo več različic

algoritmov za gradnjo odločitvenih dreves, ki za delitev množice uporabljajo različne mere informacijskih prispevkov atributov. Algoritem C4.5 [14], ki smo ga uporabljali, izhaja iz algoritma ID3 in za delitev množic uporablja informacijski prispevek.

3.2.2 Bagging

Metoda bagging združuje več odločitvenih modelov v en skupen model. To je pri klasifikacijskih problemih najlažje izvesti z glasovanjem. Različni modeli so zgrajeni na različnih, enako velikih učnih množicah, katerih primeri so izbrani iz skupne učne množice. Izbira učnih primerov za odločitvene modele poteka postopno. Prvemu modelu se naključno določi učna množica, katero vsak naslednji model prevzame in zamenja določeno število njenih primerov z novimi naključno izbranimi primeri. Tako so učne množice posameznih odločitvenih modelov različne, a še vedno temeljijo na isti osnovi.

Različni učni primeri botrujejo znatnim razlikam v zgrajenih modelih. Razlog za to je predvsem nestabilnost postopka gradnje odločitvenih dreves. Majhne razlike v učnih primerih lahko na nekem nivoju hitro spremenijo izbrani delitveni atribut, kar povzroči spremembo celotne strukture drevesa pod tem nivojem in veliko razliko v klasifikaciji. Pri glasovanju imajo vsi uporabljeni modeli enako utež in enako vplivajo na končno odločitev, ki je ponavadi točnejša od napovedi modela, zgrajenega na celotni učni množici.

3.2.3 Boosting

Metoda boosting podobno kot metoda bagging z glasovanjem združi več odločitvenih modelov, ki pa so zgrajeni tako, da drug drugega dopolnjujejo. Za razliko od metode bagging, pri kateri je vsak model zgrajen neodvisno od ostalih modelov, je pri metodi boosting vsak model odvisen od učinkovitosti predhodnega modela. Vsak naslednji odločitveni model se osredotoča na klasifikacijo primerov, ki so bili s predhodnim modelom napačno klasificirani. Pri glasovanju imajo zato točnejši odločitveni modeli večjo utež in tako večji

vpliv na končno klasifikacijo.

3.2.4 Metoda podpornih vektorjev

Metoda podpornih vektorjev (angl. support vector machine) učne primere predstavlja kot točke v prostoru, katerih položaj je določen z vrednostmi njihovih atributov. Model točke po prostoru opiše tako, da so primeri različnih razredov čim bolj ločeni. Ob klasifikaciji model testne primere umesti v isti prostor in jim glede na območje, kamor spadajo, določi razred. Točke učnih primerov, ki so najbližje ločnici med razredi, imenujemo podporni vektorji. Za vsak razred vedno obstaja vsaj en podporni vektor, pogosto pa jih je več. Za ločevanje dveh razredov potrebujemo le njune podporne vektorje, saj vsebujejo vse podatke, potrebne za določitev ločnice med njima. Vse ostale točke lahko po določitvi podpornih vektorjev zanemarimo.

3.2.5 Umetne nevronske mreže

Umetna nevronska mreža (angl. artificial neural network) je matematični model, ki temelji na bioloških nevronskih mrežah. Sestavljena je iz množice medsebojno povezanih procesnih elementov, imenovanih umetni nevroni. Nevroni so povezani z uteženimi povezavami, ki določajo obnašanje nevronske mreže. Nevroni drug drugemu pošiljajo signale. Če je vsota sprejetih signalov pri določenemu nevronu dovolj velika, se signal prenese na njegov izhod. Uteži vhodov, povezave in pragovi nevronov za prenos signalov se v fazi učenja oblikujejo in spreminjajo, dokler ne najdejo najustreznejšega odločitvenega modela.

3.2.6 Prečno preverjanje rezultatov

Pri učenju modelov lahko pride do prevelike prilagoditve učni množici (angl. overfitting), kar povzroči, da je ocena točnosti modela nenatančna in preveč optimistična. Pri osnovnem načinu preverjanja točnosti odločitvenega modela ta problem odpravimo tako, da primere razdelimo na učno in testno

množico. Učno množico uporabimo za učenje modela, na testni množici pa preverjamo njegovo točnost. Če imamo na voljo majhno število primerov, je takšno ocenjevanje točnosti neustrezno, saj zastopanost razredov v učni in testni množici ni vedno ustrezna.

Tovrstne težave odpravimo s prečnim preverjanjem. Pri prečnem preverjanju klasifikacijske točnosti celotno množico primerov razdelimo na n skupin. Primere $n - 1$ skupin uporabimo za gradnjo odločitvenega modela, preostala skupina pa služi kot testna množica, ki pokaže, kako točen je zgrajeni odločitveni model. Postopek ponovimo n-krat. V vsaki ponovitvi testno množico predstavlja druga skupina primerov. Tako dobimo n klasifikacijskih točnosti, katerih povprečje predstavlja oceno točnosti končnega drevesa, ki je zgrajen iz vseh učnih primerov. V praksi je najbolj priporočeno 10-kratno prečno preverjanje.

Poglavje 4

Nadzor kakovosti komutatorjev

Za namene avtomatskega nadzora kakovosti komutatorjev smo implementirali postopek, ki obsega dve fazi. V prvi fazi iz slike pridobimo potrebne atribute, v drugi pa komutatorju na podlagi pridobljenih atributov določimo razred, kateremu pripada. To poglavje predstavlja uporabljena orodja, postopek nadzora kakovosti in njegovo implementacijo.

4.1 Orodja in programske knjižnice

4.1.1 Visual Studio 2010

Za izvedbo vgradnega sistema za nadzor kakovosti komutatorjev smo uporabljali Microsoftovo razvojno okolje Visual Studio 2010 [15], ki je namenjeno širokemu obsegu uporabnikov. Osnovna različica podpira programske jezike C/C++, Visual Basic, C# in F#. Visual Studio smo večinoma uporabljali v navezi s programskim jezikom C++ in programskim paketom OCL.

4.1.2 Programski paket OCL

Visual Studio smo povezali s programskim paketom OCL [16], ki je del knjižnice OpenCV (angl. Open Computer Vision) in implementira nekatere njene funkcije. OpenCV [6] je programska knjižnica, ki implementira množico

funkcij strojnega vida od štetja pik do zahtevnejših algoritmov za prepoznavanje lastnosti slik. Programski paket OCL za razliko od osnovne programske knjižnice OpenCV funkcije strojnega vida izvaja paralelno s pomočjo ogrodja OpenCL (angl. Open Computing Language) [7]. Za potrebe projekta COPCAMS so ustrezne le funkcije iz programskega paketa OCL, saj je glavni cilj projekta paralelizacija postopkov s pomočjo platforme STHORM [2].

4.1.3 Weka

Weka [5] je odprtokodna programska oprema, ki nudi širok nabor orodij in algoritmov za analizo podatkov ter strojno učenje. Napisana je v javi, kar zagotavlja dobro prenosljivost med različnimi platformami. Weka podpira standardne funkcije podatkovnega rudarjenja, kot so predpriprava podatkov, združevanje v skupine, klasifikacija, regresija, vizualizacija in izbira atributov. Prilagodljiva je potrebam uporabnika, saj se lahko uporablja kot grafični vmesnik, ki ne zahteva znanja programiranja, ali pa kot programska knjižnica v povezavi z razvojnim okoljem. V okviru diplomskega dela je bila Weka uporabljena za gradnjo odločitvenih modelov in preverjanje njihove točnosti.

4.1.4 Eclipse

Eclipse [17] je programsko okolje, ki je primarno namenjeno programiranju v javi, omogoča pa tudi programiranje v drugih programskih jezikih. Eclipse smo v povezavi z Weko uporabljali v zadnji fazi strojnega učenja, katere glavni del je bila implementacija in testiranje metaklasifikatorja.

4.2 Pridobivanje atributov iz slike

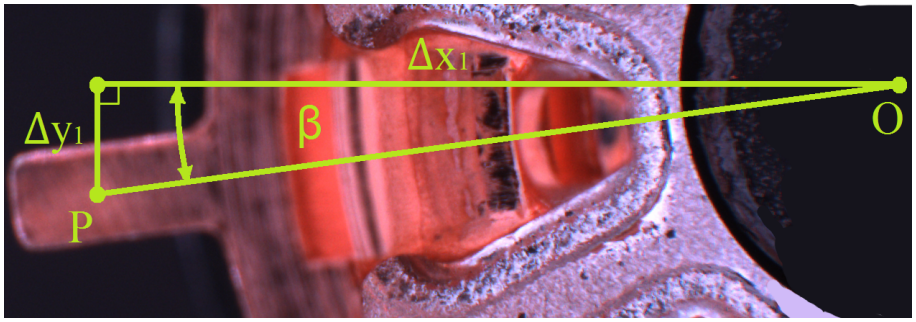
Atribute iz slike pridobivamo v več korakih. V prvem koraku določimo kot nagiba segmenta na sliki, v drugem koraku iz slike izrežemo določena območja, v tretjem koraku pa iz izrezanih območij pridobimo atribute. Koraki postopka pridobivanja atributov iz slike so opisani v naslednjih razdelkih.

4.2.1 Določanje kota nagiba segmenta na sliki

Slike segmentov komutatorja so zajete s pomočjo za to razvitega manipulatorja. Čeprav manipulator zagotovi približno poravnanost segmentov, še vedno pride do rahlih odstopanj. Ta sicer niso večja od nekaj milimetrov, a vseeno onemogočajo pravilen izrez pomembnih območij. Prvi korak pri pridobivanju lastnosti s pomočjo strojnega vida je zato določanje kota, za katerega je segment na sliki zavrten. Za izračun kota je potrebno določiti položaj dveh točk na segmentu komutatorja. To sta središče luknje komutatorja (točka O na sliki 4.1) in točka na sredini nožice (točka P na sliki 4.1).

Iz zajete slike najprej pridobimo rdeči barvni kanal, ki se je pri iskanju točk izkazal za najustreznejšega. Pridobljeni kanal predstavlja intenzitetno sliko, ki jo z upragovanjem spremenimo v binarno. To sliko nato obdelamo z morfološkima operacijama odprtja in zaprtja ter tako odpravimo morebitne šume in nepravilnosti na sliki.

Binarno sliko uporabimo za iskanje potrebnih točk. Najprej poiščemo središče luknje komutatorja. Polmer kroga je konstanten, zato smo se odločili, da bomo središče kroga poiskali s postopkom iskanja predloge na sliki. Postopek pove, kateri del slike se po intenziteti najbolj ujema s predlogo, vidno na sliki 4.2a. Predloga predstavlja del kroga, ki je skupen vsem fotografijam ne glede na njihovo zavrtenost. Relacije med točkami pri iskanju središča



Slika 4.1: Shema postopka za določanje nagiba segmenta komutatorja

kroga so prikazane na sliki 4.2b. Predloga se najbolje ujema z območjem, označenim z zelenim pravokotnikom. Z metodo iskanja predloge na sliki torej dobimo koordinati točke M. Središče kroga, ki je predstavljeno s točko O, izračunamo z upoštevanjem relativne razdalje do točke M, ki je v vseh primerih enaka. Koordinatam točke M po osi x prištejemo Δx_2, po osi y pa Δy_2.

Ko najdemo središče luknje, poiščemo še točko na sredini nožice (točka P na sliki 4.3). Koordinato x točke P dobimo tako, da od koordinate x središča luknje odštejemo Δx_1. Na tem mestu pregledamo celoten stolpec slike, ki je na sliki 4.3 predstavljen z zeleno črto, in poiščemo najvišje ležeči bel slikovni element. Točka T, kjer se nahaja ta element, predstavlja vrh nožice. Koordinati y točke T prištejemo polovico števila belih elementov v stolpcu, kar znaša polovico višine nožice in je na sliki 4.3 označeno z Δy_3. Tako dobimo višino sredine nožice in s tem koordinato y točke P.

S pomočjo dobljenih točk lahko izračunamo kot β, za katerega je segment na sliki zavrten, z enačbo:

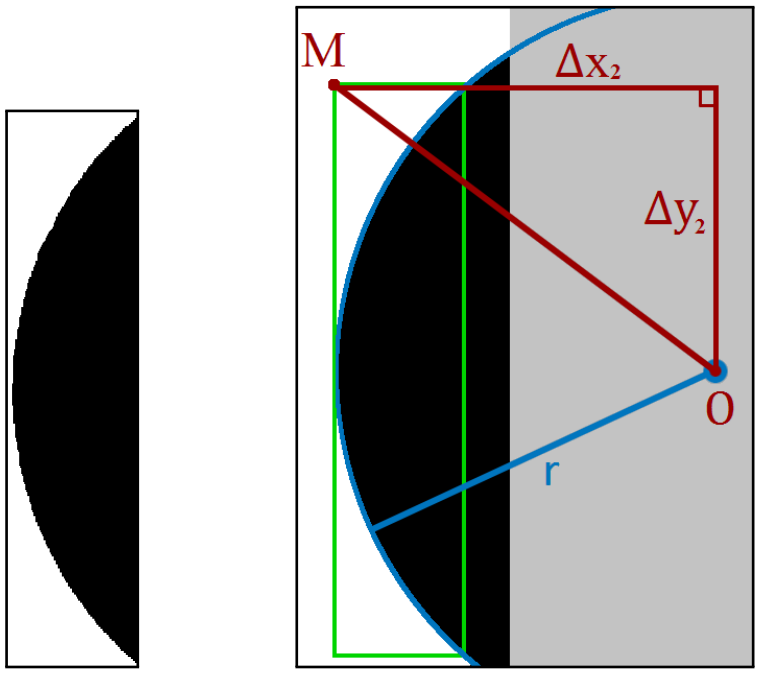
$$\beta = -\arctan\frac{\Delta y_1}{\Delta x_1}, \tag{4.1}$$

kjer je Δx_1 razlika med koordinatama x točk O in P, Δy_1 pa razlika med koordinatama y istih točk.

4.2.2 Izrezovanje območij

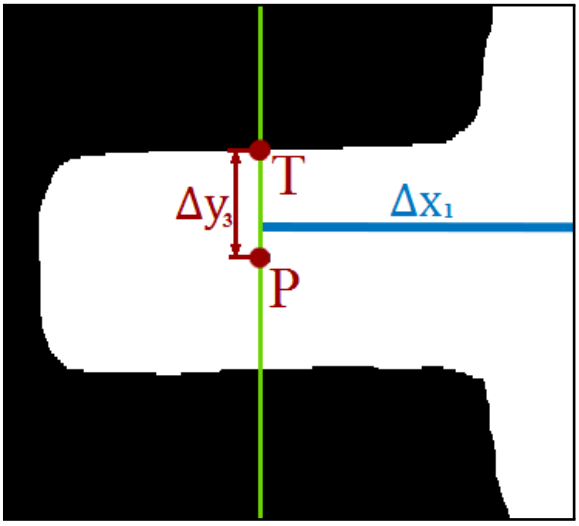
Pri preverjanju kakovosti komutatorja iščemo štiri vrste napak, katerih pokazatelji so prisotni v točno določenih območjih slike. Ker bi ostala območja slike predstavljala šum v podatkih, jih z uporabo binarnih mask, prikazanih na sliki 4.4, odstranimo. Pri tem je izbira maske odvisna od iskane vrste napake.

Izrez teh območij smo sprva izvedli tako, da smo slike z vrtenjem in premikom poravnali tako, da so bila območja vedno na istem mestu in je bil posledično njihov izrez preprost. Problem pri tem postopku je predstavljalo glajenje, ki je ob vrtenju popačilo sliko. Ta problem smo odpravili tako, da

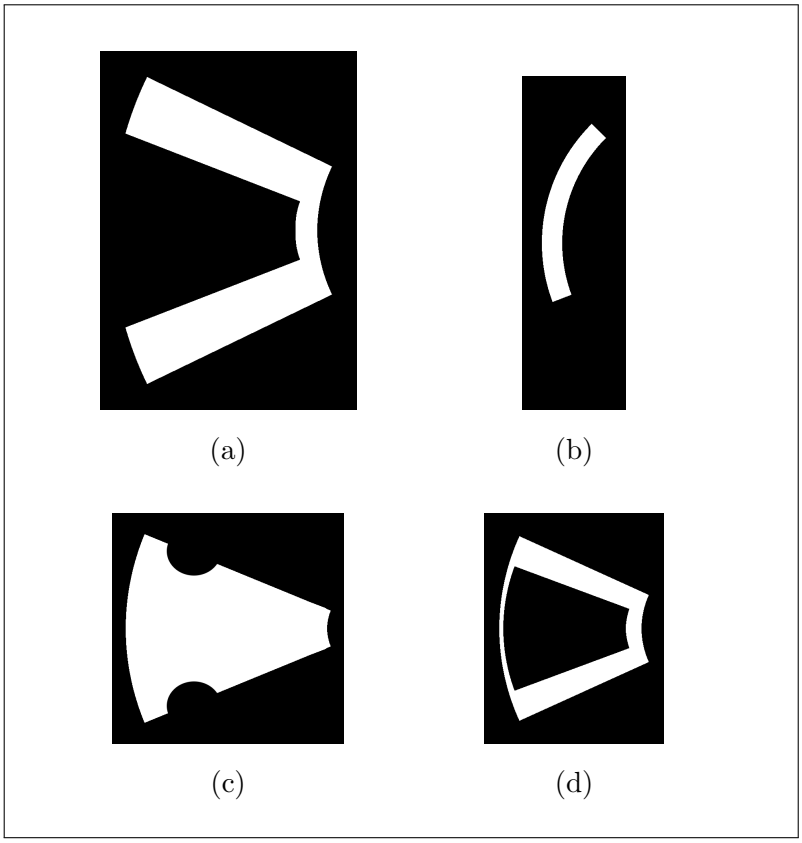


(a) (b)

Slika 4.2: Iskanje točke M: a) predloga za iskanje točke M, b) relacija med točko M in središčem luknje O.



Slika 4.3: Iskanje točke P na sredini nožice komutatorja.



Slika 4.4: Primeri binarnih mask, ki služijo za izrez območij, kjer se pojavljajo napake: a) poškodba metalizacije, b) neorientiranost, c) presežek spojne paste in d) primanjkljaj spojne paste.

smo položaj in kot rotacije mask prilagodili posamezni sliki. Posledično ni potrebe po rotiranju slike in le-te zato ne popačimo.

Za pravilno postavitev maske so potrebni trije podatki. To so polmer oddaljenosti središča območja za izrez od središča luknje (r_{obm} na sliki 4.5), koordinate središča luknje in kot, za katerega je segment na sliki zavrten. Prvi podatek je konstanten za vsako območje posebej in ga ni potrebno izračunati, druga dva pa smo izračunali, kot je opisano v razdelku 4.2.1. Izračunani kot uporabimo za vrtenje maske okrog njene osi. Tako dosežemo, da je maska pravilno orientirana glede na komutator na sliki.

Kot vidimo na sliki 4.5, koordinato središča maske, ki je predstavljena s točko S, dobimo tako, da koordinati x središča luknje (točka O) odštejemo razdaljo a, koordinati y pa prištejemo razdaljo b. Vrednosti a in b sta:

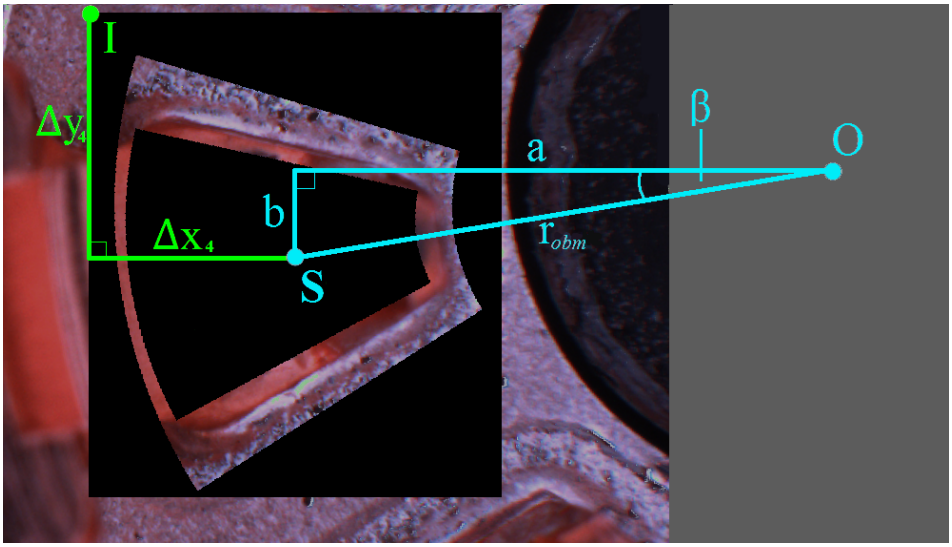
$$a = r_{obm} \cos \beta, \tag{4.2}$$

$$b = r_{obm} \sin \beta. \tag{4.3}$$

Ko najdemo središče maske, ga uporabimo za izračun koordinat levega zgornjega kota maske, ki je na sliki predstavljen s točko I. Točko izračunamo tako, da koordinati x točke S odštejemo Δx_4, kar predstavlja polovico širine binarne maske, koordinati y pa odštejemo Δy_4, kar je polovica višine binarne maske. Ko imamo točko I, iz slike izrežemo pravokotnik, katerega levi zgornji kot se nahaja v točki I in je enake velikosti kot binarna maska. Nadaljnji postopek se izvaja samo na izrezanem pravokotniku, katerega primer je prikazan na sliki 4.6a, in je skladen z uporabljeno masko, ki jo vidimo na sliki 4.6b. Preostanka slike v nadaljevanju ne obravnavamo.

V zadnjem koraku na izrezanem pravokotniku zatemnimo nepomembno območje določeno z binarno masko in pridemo do rezultata prikazanega na sliki 4.6c. To storimo tako, da za vsak slikovni element na sliki izračunamo novo vrednost po enačbi:

$$V'_{(u,v)} = \begin{cases} 0 & \text{če velja } M_{(u,v)} = 0, \\ V_{(u,v)} & \text{sicer,} \end{cases} \tag{4.4}$$

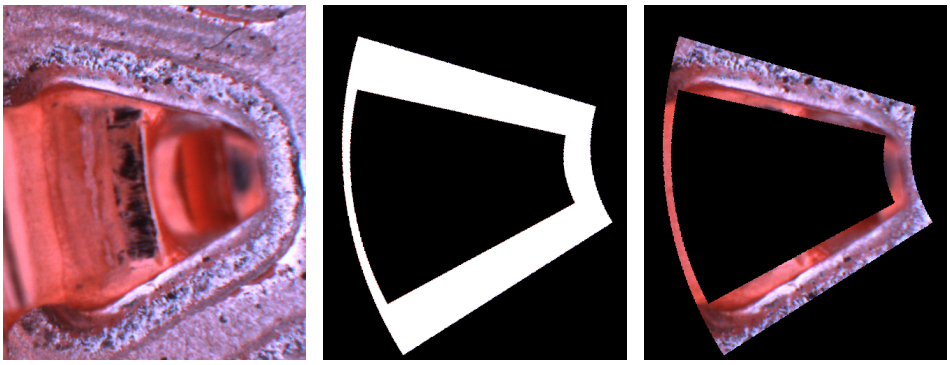


Slika 4.5: Določanje območja za izrez.

kjer sta spremenljivki u in v koordinati elementa, $V'_{(u,v)}$ nova vrednost elementa slike, $V_{(u,v)}$ stara vrednost elementa slike, $M_{(u,v)}$ pa vrednost istoležnega elementa v binarni maski. Postopek ohrani le del slike, na katerem imajo istoležni slikovni elementi v binarni maski vrednost večjo od nič.

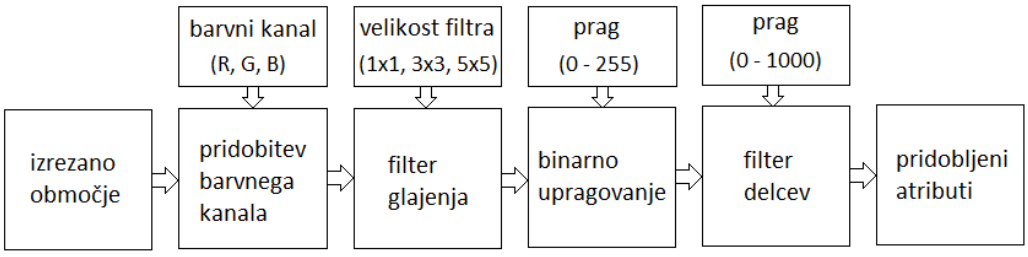
4.2.3 Pridobivanje atributov

Ko iz slike izrežemo določeno območje, s postopkom prikazanim na sliki 4.7 izračunamo atribute. Vhodne spremenljivke postopka so barvni kanal, velikost filtra glajenja, prag binarnega upragovanja in prag filtra delcev. Vhodne spremenljivke so bile določene s predhodno optimizacijo postopka v programu LabVIEW. Najprej iz slike pridobimo barvni kanal, pri čemer je izbira kanala odvisna od iskane vrste napake oziroma od območja, ki ga preiskujemo. Tako dobimo intenzitetno sliko enega od kanalov. Dobljeno sliko, katere primer vidimo na sliki 4.8a, nato s filtrom mediane zgladimo. Na zglajeni sliki nato izvedemo postopek upragovanja, ki slikovne elemente manjše od praga postavi na 0, ostale pa na 1. Tako dobimo binarno sliko, katere primer vidimo na sliki 4.8b.

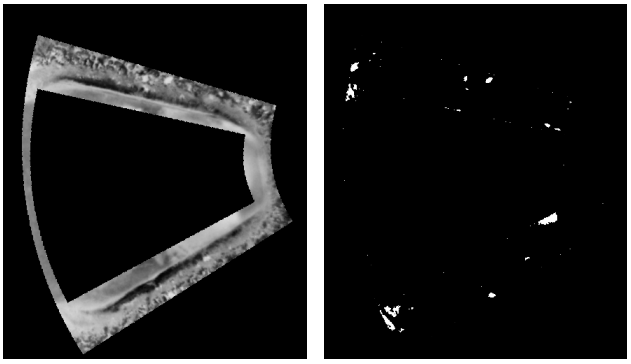


(a) (b) (c)

Slika 4.6: Obdelava izrezanega dela slike komutatorja: a) izrezani del slike, b) binarna maska, c) rezultat zatemnitve izrezanega dela slike z binarno masko po enačbi (4.4)



Slika 4.7: Shema postopka pridobivanja atributov iz izrezanega območja.



(a) (b)

Slika 4.8: Intenzitetna slika in njeno upragovanje: a) primer intenzitetne slike rdečega kanala, b) rezultat njenega binarnega upragovanja.

Naslednji korak je filtriranje majhnih delcev. Preden lahko na sliki uporabimo filter delcev in izločimo premajhne, je treba delce najti. Posamezen delec predstavlja skupina povezanih slikovnih elementov ospredja na sliki. Binarna slika ima vse slikovne elemente ospredja označene z 1. Da lahko ločimo posamezne delce med sabo, jih moramo povezati v skupine tako, da vse elemente ene skupine označimo z enako oznako, ki pa je večja od 0. Za to je najprimernejši postopek označevanja območij, ki pa v programskem paketu OCL ni implementiran, zato smo ga na podlagi psevdokode [18] implementirali z uporabo programskega ogrodja OpenCL.

Postopek označevanja območij vrne sliko, na kateri so elementi posameznega območja označeni z enako pozitivno številko, elementi ozadja so označeni z 0, luknje v delcih pa nosijo vrednost –1. Za lažje in hitrejše pridobivanje atributov smo z ogrodjem OpenCL implementirali še funkcijo, ki prešteje število slikovnih elementov, ki pripadajo določeni skupini, in izdela histogram slike. Histogram je enodimenzionalna tabela, v kateri vsak stolpec hrani število slikovnih elementov, ki imajo vrednost enako indeksu stolpca. V histogram ne zapisujemo slikovnih elementov z vrednostma –1 in 0, torej lukenj v delcih in ozadja.

Filter delcev zanemari delce, ki so manjši od določene meje, tako da

odstrani delce, ki vsebujejo premalo slikovnih elementov. Vsak stolpec histograma predstavlja delec in hrani število slikovnih elementov, ki mu pripadajo. Filter delcev torej zbriše stolpce, katerih vrednost je manjša od zahtevane. Tako dosežemo, da histogram hrani samo še dovolj velike delce in njihove velikosti.

Na podlagi pridobljene slike in histograma lahko pridobimo naslednje atribute:

- število delcev – število stolpcev v tabeli histograma, katerih vrednost je večja od 0;
- vsota delcev – seštevek vrednosti vseh stolpcev v tabeli histograma;
- velikost največjega delca – največja vrednost v tabeli histograma;
- velikost najmanjšega delca – najmanjša vrednost v tabeli histograma;
- delež vseh lukenj – število slikovnih elementov z vrednostjo –1 delimo s številom vseh slikovnih elementov z vrednostjo različno od 0;
- delež lukenj v največjem elementu – ker so vse luknje na sliki označene z –1 in ne vemo, katera pripada največjemu delcu, moramo le-tega pridobiti iz slike in samo na njem ponoviti celoten postopek označevanja delcev. Izračun atributa je nato enak izračunu deleža vseh lukenj.

Postopek pridobivanja atributov iz slik je tako zaključen. Atribute štirih območij pridobimo iz vseh slik učne množice in jih zapišemo v datoteko formata ARFF, ki jo v fazi strojnega učenja uporabimo v orodju Weka. Datoteka poleg atributov za vsako sliko hrani tudi kakovostni razred, ki mu komutator na sliki pripada in je bil za vsako sliko določen ročno s strani strokovnjakov. Razred je enak vrsti napake na komutatorju oz. “brez napak”.

4.3 Klasifikacija

Klasifikacijo smo razdelili na dva nivoja. Spodnji nivo sestavljajo štirje osnovni klasifikatorji. Vsak izmed njih je zadolžen za prepoznavanje ene

od štirih vrst napak. Na zgornjem nivoju imamo metaklasifikator, ki napovedi klasifikatorjev na spodnjem nivoju združi in določi, v kateri razred spada komutator. Za klasifikatorje spodnjega nivoja smo uporabili odločitvena drevesa, zgrajena z algoritmom J48, ki predstavlja Wekino implementacijo algoritma C4.5 v javi. Odločitveno drevo je za naše potrebe najprimernejše, saj ga je najlažje implementirati na nivoju končne industrijske aplikacije. Končna aplikacija bo klasifikatorje brala iz posameznih datotek, kar omogoča enostavno spreminjanje odločitvenih modelov tudi po končani fazi razvoja.

Zgrajeno odločitveno drevo Weka izpiše kot strukturirano besedilo, katerega primer vidimo na sliki 4.9. Rdeče vrednosti predstavljajo indeks atributa, ki ga v pogoju primerjamo z modro obarvano vrednostjo, zelene vrednosti pa predstavljajo razred, ki ga klasifikator vrne za obravnavani primer. Konkretno vrednost 1 pomeni, da je napaka prisotna, vrednost 0 pa, da napake ni. V oklepaju je predstavljena porazdelitev po razredih, a je v trenutni izvedbi ne upoštevamo.

Besedilo vsakega od štirih odločitvenih dreves na spodnjem nivoju zapišemo v tekstovno datoteko, ki jo nato v aplikaciji preberemo, razčlenimo z razčlenjevalno funkcijo in uporabimo za klasifikacijo posameznih primerov. Psevdokodo razčlenjevalne funkcije vidimo na sliki 4.10.

```
1 <= 4155
|   2 <= 188: 1 (36.0/1.0)
|   2 > 188: 0 (12.0/1.0)
1 > 4155
|   3 <= 2588
|   |   0 <= 2439: 1 (5.0)
|   |   0 > 2439:
|   |   |   4 <= 2008: 0 (11.0/2.0)
|   |   |   4 > 2008: 1 (9.0/1.0)
|   3 > 2588: 1 (9.0)
```

Slika 4.9: Izpis odločitvenega drevesa, dobljen z orodjem Weka.

```
globina = -1
for vrstica in tekstovna_datoteka:
    if globina == -1
        if preveri_pogoj()
            if  vsebuje_dvopičje()
                return razred_vrstice()
        else
            globina = globina_vrstice();
    else if globina == globina_vrstice()
        if  vsebuje_dvopičje()
            return razred_vrstice()
    globina = -1;
```

Slika 4.10: Psevdokoda razčlenjevalne funkcije za branje zapisov odločitvenih dreves.

V psevdokodi so uporabljene naslednje funkcije:

- preveri_pogoj() – preveri pogoj trenutne vrstice in vrne 1, če velja, in 0 sicer;
- vsebuje_dvopičje() – preveri ali vrstica vsebuje dvopičje in vrne 1, če velja, in 0 sicer;
- razred_vrstice() – vrne vrednost razreda v vrstici;
- globina_vrstice() – vrne globino vrstice.

Razčlenjevanje se začne v prvi vrstici tekstovne datoteke. Če pogoj velja, preverimo, ali vrstica predstavlja list ali vozlišče odločitvenega drevesa. Če vrstica vsebuje dvopičje, predstavlja list, saj za dvopičjem vedno najdemo razred. Tega nato vrnemo in tako zaključimo iskanje. Če vrstica ne vsebuje dvopičja, predstavlja vozlišče drevesa. Ker smo že prej ugotovili, da pogoj v vozlišču velja, zaključimo prvi prehod zanke in se pomaknemo vrstico nižje, kjer se nahaja naslednji pogoj. Če prvi pogoj ne velja, moramo najti njegovo

alternativo. To storimo tako, da najdemo naslednjo vrstico, katere globina je enaka globini prvega pogoja. Pogoj v najdeni vrstici zagotovo velja, zato ga ni potrebno preverjati. Preverimo le, ali je vrstica list ali vozlišče drevesa in glede na ugotovljeno nadaljujemo po zgornjem vzorcu. Na ta način iz štirih tekstovnih datotek preberemo štiri klasifikatorje, s katerimi iščemo prisotnost štirih vrst napak. Rezultate klasifikatorjev na spodnjem nivoju v končno klasifikacijo združi metaklasifikator, ki deluje na naslednji način. Če nobeden od klasifikatorjev ne najde napake, je komutator dober (brez napak), če napako najde vsaj eden, pa komutator ni ustrezen. Vrsta napake je v primeru konfliktov med klasifikatorji določena z upoštevanjem prioritet klasifikatorjev. Če za določen primer napako javi več klasifikatorjev, se torej upošteva klasifikator z najvišjo prioriteto med njimi.

Poglavje 5

Izračuni in rezultati

To poglavje opisuje pridobivanje rezultatov in njihovo ovrednotenje. Predstavljeni so učna množica in načini uporabe učnih primerov za učenje klasifikatorjev. Testirali smo štiri dodatne metode za gradnjo odločitvenih modelov, katerih rezultati so primerjani z rezultati odločitvenih dreves.

5.1 Učna množica

Učna množica vsebuje 359 učnih primerov, katerih porazdelitev po razredih vidimo v tabeli 5.1. Učno množico predstavljajo slike komutatorjev, razdeljene v pet razredov. Prvi razred predstavlja komutatorje brez napak, ki so primerni za nadaljnjo proizvodnjo, ostali štirje razredi pa predstavljajo neustrezne komutatorje. Uporabnik (Kolektor Group d.o.o.) je izrazil željo po klasifikaciji v vseh pet razredov, ne le v razreda dober in slab, saj to omogoča natančnejše spremljanje proizvodnje in lažje odpravljanje napak. Iz tega razloga smo se osredotočili zgolj na ta petrazredni klasifikacijski problem.

Komutatorji z napako so razdeljeni glede na vrsto napake, ki je razvidna iz slike. Vrste napak so opisane v podpoglavju 2.3. Za vsako sliko v učni množici datoteka ARFF hrani pridobljene atribute vseh štirih območij in razred, ki mu slika pripada in je bil določen s strani ekspertov. Vsaka vrstica datoteke hrani podatke za posamezno sliko. Podatki v vrsticah so zapisani

Tabela 5.1: Porazdelitev komutatorjev po razredih v učni množici.

Razred	Število primerov
brez napak	212
poškodba metalizacije	35
primanjkljaj spojne paste	49
presežek spojne paste	35
neorientiranost	32

po skupinah, pri čemer vsaka skupina hrani po šest atributov določenega območja, zadnji element pa predstavlja razred slike.

5.2 Načini učenja

Glede na učno množico in nabor atributov smo se odločili, da bomo testirali pet načinov učenja klasifikatorjev na spodnjem nivoju. Grafični prikaz naslednjih načinov učenja vidimo na sliki 5.1.

- Način 1 – Pri gradnji klasifikatorjev na spodnjem nivoju za iskanje posamezne vrste napake uporabimo samo šest atributov, ki so pridobljeni iz območja, na katerem je ta napaka najbolje vidna. Pri tem učno množico predstavljajo samo dobri primeri in primeri iskane vrste napake. Primere z ostalimi vrstami napak pri učenju zanemarimo. Pri tem načinu predpostavimo, da atributi, ki niso del kritičnega območja, pri iskanju določene vrste napake vnašajo šum in tako znižujejo točnost. Prav tako predpostavimo, da šum med podatke vnašajo primeri drugih vrst napak v učni množici, zato jih iz nje odstranimo.

- Način 2 – Pri gradnji klasifikatorjev na spodnjem nivoju za iskanje posamezne vrste napake uporabimo vse atribute vseh območij. Pri tem učno množico predstavljajo samo primeri brez napak in primeri iskane

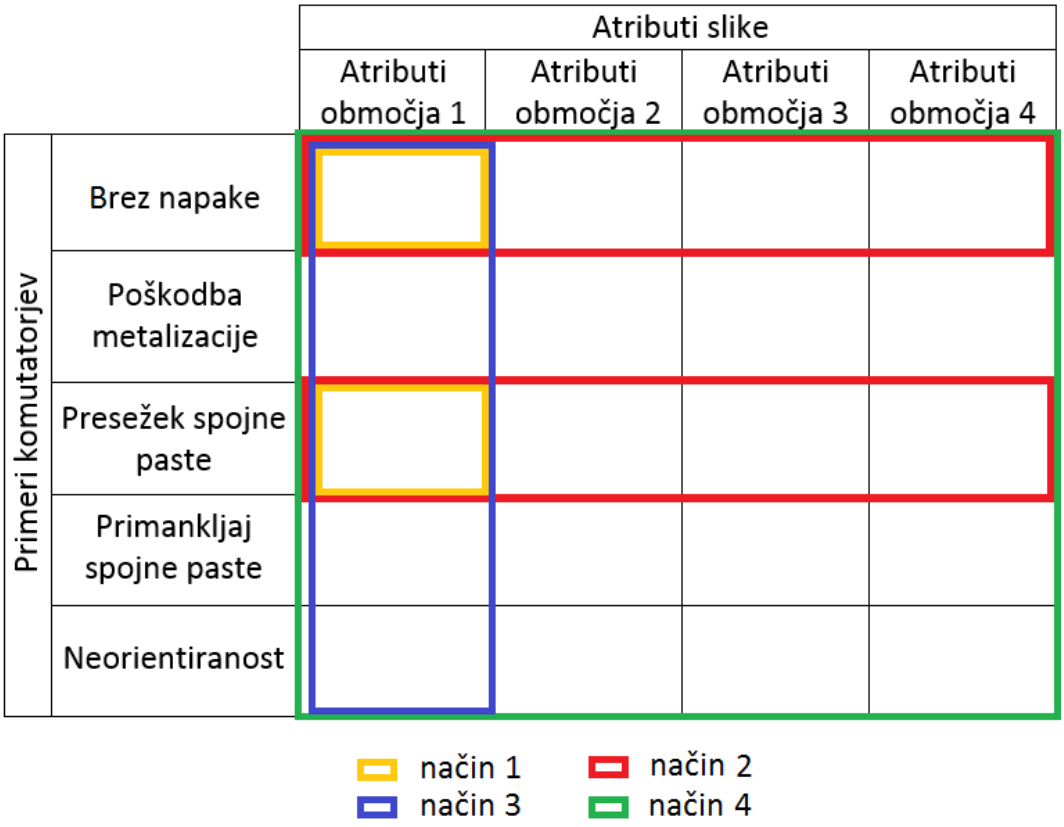
vrste napake. Primere z ostalimi vrstami napak pri učenju zanemarimo. Pri tem načinu izbiro atributov popolnoma prepustimo klasifikatorju. Kot pri načinu 1 predpostavimo, da šum v podatke vnašajo primeri drugih vrst napak v učni množici, zato jih iz nje odstranimo.

- Način 3 – Pri gradnji klasifikatorjev na spodnjem nivoju za iskanje posamezne vrste napake uporabimo samo šest atributov, ki so pridobljeni iz območja, v katerem je ta vrsta napake (najbolje) vidna. Pri tem v učni množici dobre primere predstavljajo vsi primeri, ki ne vsebujejo iskane vrste napake, slabe primere pa primeri, ki jo vsebujejo. Pri tem načinu predpostavimo, da atributi, ki niso del kritičnega območja, pri iskanju določene vrste napake vnašajo šum in tako znižujejo točnost. Za razliko od načina 1 primerov, ki nimajo iskane vrste napake, ne zavržemo in tako ohranimo vse podatke, kar utegne izboljšati točnost.

- Način 4 – Pri gradnji klasifikatorjev na spodnjem nivoju za iskanje posamezne vrste napake uporabimo vse atribute vseh območij. Pri tem v učni množici dobre primere predstavljajo vsi primeri, ki ne vsebujejo iskane vrste napake, slabe primere pa primeri, ki jo vsebujejo. Pri tem načinu izbiro atributov popolnoma prepustimo klasifikatorju, primerov, ki nimajo iskane vrste napake, pa ne zavržemo in tako ohranimo vse podatke.

- Način 5 – Učenja ne opravimo na dveh nivojih s štirimi klasifikatorji na spodnjem nivoju in enem na zgornjem, temveč vse pridobljene podatke uporabimo za učenje enega večrazrednega klasifikatorja, ki primere klasificira v enega od petih razredov. Primerjava med rezultati tega načina in rezultati ostalih načinov bo pokazala, če je klasifikacija v dveh nivojih sploh smiselna, in če je, kolikšne so izboljšave.

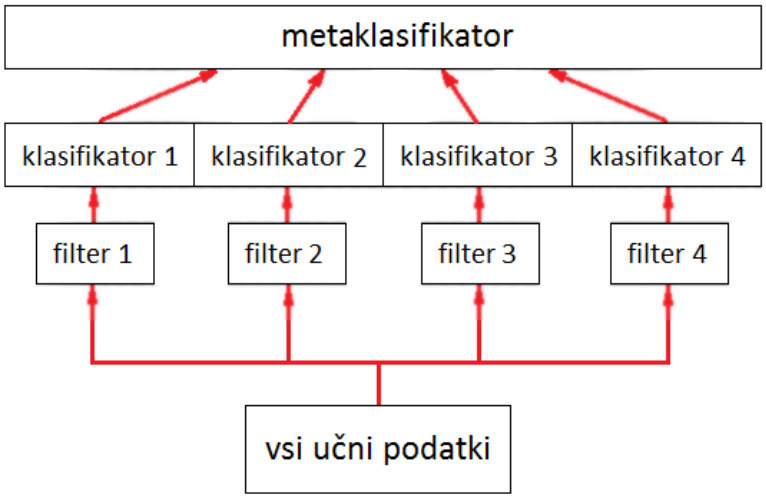
Da bi dobili čim točnejše rezultate učenja, smo njihovo točnost merili s pomočjo prečnega preverjanja. Za preverjanje točnosti klasifikacije je bila



Slika 5.1: Testirani načini učenja. Barvni okviri predstavljajo obseg učnih primerov, uporabljenih pri posameznem načinu učenja.

potrebna implementacija metaklasifikatorja, na katerem je možno izvajanje prečnega preverjanja, ki ga ponuja orodje Weka, kar zagotovi ustreznejše ocenjevanje točnosti klasifikacije.

Metaklasifikator (glej sliko 5.2) na spodnjem nivoju vsebuje tabelo štirih klasifikatorjev. V učni funkciji metaklasifikatorja poteka učenje klasifikatorjev spodnjega nivoja. Vsak tak klasifikator za učenje prejme ustrezno filtrirane podatke, ki so odvisni od iskane vrste napake in izbranega načina učenja. Ob klasifikaciji testnega primera metaklasifikator pridobi napovedani razred od vsakega od klasifikatorjev spodnjega nivoja in se nato glede na odgovore ter določene prioritete odloči, kakšna bo končna klasifikacija primera. Poleg nastavitve načina učenja in prioritet klasifikatorjev lahko nastavljamo tudi metodo učenja osnovnih klasifikatorjev in tako primerjamo točnost odločitvenih dreves s točnostjo drugih metod.



Slika 5.2: Shema metaklasifikatorja.

5.3 Primerjava načinov in metod učenja

Za gradnjo klasifikatorjev smo uporabili pet metod učenja, ki jih ponuja orodje Weka:

- J48 – Wekina implementacija algoritma C4.5, ki generira odločitveno drevo,
- ANN – Wekina metoda učenja z umetno nevronsko mrežo Multilayer Perceptron,
- Bagging – Wekina različica metode bagging,
- AdaBoost – Wekina različica metode boosting,
- SMO – Wekina različica metode podpornih vektorjev.

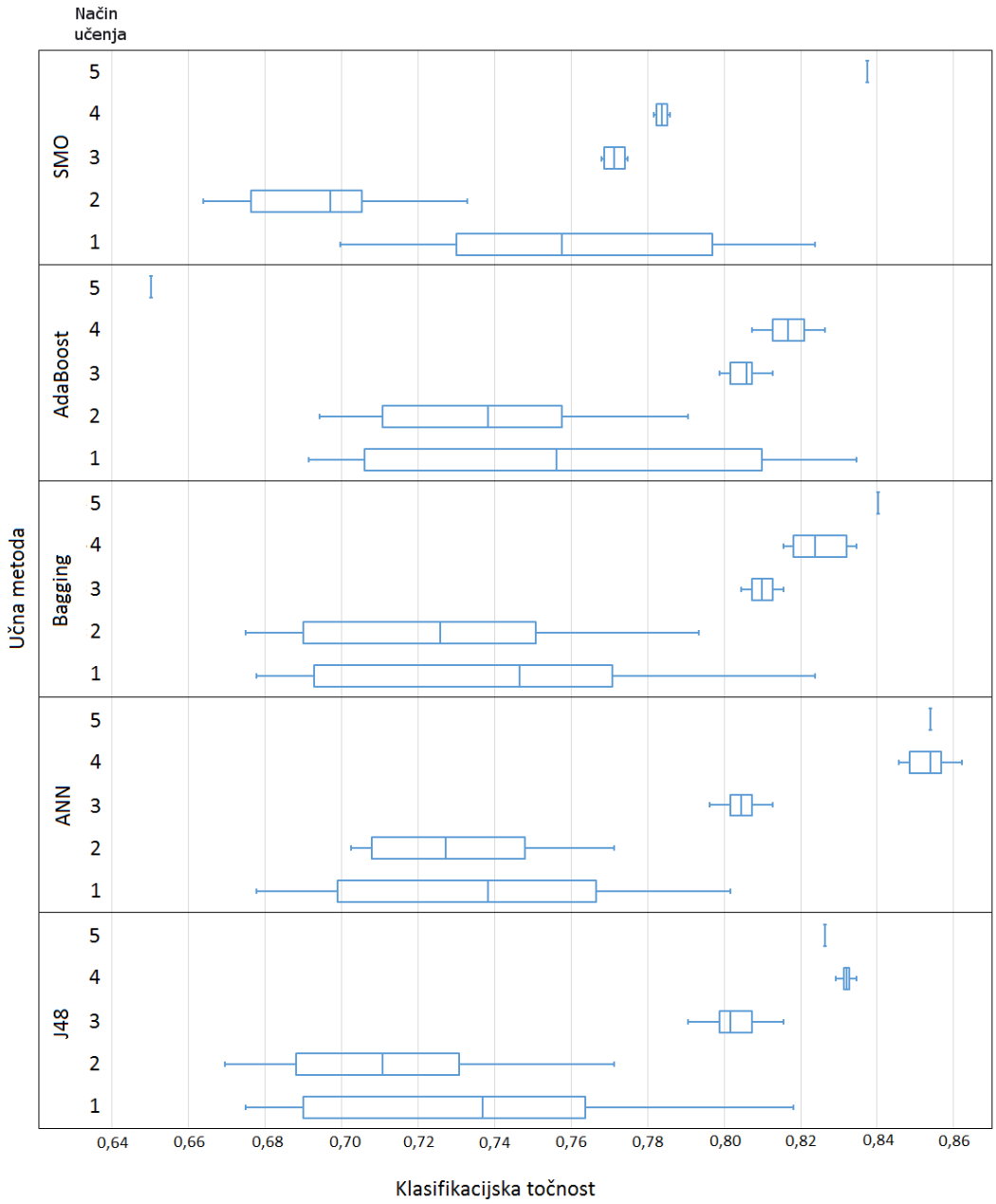
Na vsaki od teh metod smo testirali pet načinov učenja, ki so opisani v podpoglavju 5.1. Rezultate prečnega preverjanja vidimo na sliki 5.3. Slika prikazuje porazdelitev klasifikacijskih točnosti, ki jih dosega posamezna kombinacija metode in načina učenja ob različnih kombinacijah prioritet klasifikatorjev na spodnjem nivoju, določenih v metaklasifikatorju. Širina porazdelitve klasifikacijske točnosti pove, koliko je klasifikacijska točnost posamezne kombinacije odvisna od prioritete klasifikatorjev na spodnjem nivoju. Pri načinu učenja 5 prioritet ne določamo, zato so njegovi rezultati na sliki predstavljeni kot posamezne vrednosti. Na sliki 5.3 vidimo, da pri različnih načinih učenja dobimo podobno porazdelitev ne glede na uporabljeno metodo. Točnost načinov 1 in 2 je pri vseh metodah zelo odvisna od nastavljene prioritete in se lahko razlikuje za skoraj 15 odstotnih točk, medtem ko sta načina 3 in 4 od nastavljene prioritete veliko manj odvisna.

Skupna lastnost načinov 1 in 2 je njuna učna množica. Sestavljajo jo samo primeri brez napak in primeri iskane vrste napake. Primeri ostalih vrst napak so odstranjeni in tako ne nastopajo pri gradnji klasifikatorja. Podrobnejši pregled rezultatov načinov 1 in 2 je pokazal, da vsi osnovni klasifikatorji dobro klasificirajo komutatorje brez napak, saj slednji sestavljajo učno množico

vsakega od klasifikatorjev. Drugače je pri komutatorjih z napako, pri katerih so rezultati posameznih osnovnih klasifikatorjev zelo različni. Nekateri so pri napovedovanju vrste napake veliko bolj nagnjeni k pozitivnemu rezultatu in tako ob napačno nastavljenih prioritetah preglasujejo tiste, ki so manj nagnjeni k pozitivnemu rezultatu, a hkrati bolj točni. To je glavni razlog za velik vpliv prioritet na klasifikacijsko točnost. S pravilno nastavljenimi prioritetami sta lahko načina 1 in 2 konkurenčna ali celo boljša od preostalih načinov. Za razliko od načinov 1 in 2 načina 3 in 4 za učenje uporabljata vse primere komutatorjev, kar povzroči manjšo odvisnost klasifikacijske točnosti od nastavljenih prioritet. Ker učne množice sestavljajo vsi primeri komutatorjev, je pri klasifikaciji veliko manj konfliktov med osnovnimi klasifikatorji. Prioriteta te redke konflikte razreši in zato še vedno nekoliko vpliva na klasifikacijsko točnost.

Na klasifikacijsko točnost vpliva tudi izbira atributov. Pri načinih 1 in 2, pri katerih učno množico sestavljajo samo primeri brez napak in primeri iskane vrste napake, se bolj izkaže način 1, ki za učenje uporablja samo šest atributov, izmed načinov 3 in 4 pa je boljši način 4, ki uporablja vse atribute. Kot vemo iz razdelka 4.2.2, atribute pridobivamo iz območij, ki vsebujejo glavne pokazatelje določenih vrst napak. Ko pri načinu 2 uporabimo atribute vseh območij, hkrati pa učno množico sestavljajo le primeri ene vrste napake, atributi nepotrebnih območij predstavljajo šum in nižajo klasifikacijsko točnost. Nasprotni učinek dosežemo pri načinu 4, pri katerem učno množico sestavljajo primeri vseh vrst napak. Atributi vseh območij v tem primeru ne predstavljajo šuma, temveč služijo za razločevanje med posameznimi napakami.

V primerjavi z načinom 5, ki predstavlja enonivojsko klasifikacijo z vsemi učnimi primeri in vsemi atributi, smo izboljšavo dosegli pri metodah ANN, J48 in AdaBoost. Pri prvih dveh se je nekoliko bolj izkazal način 4, pri metodi AdaBoost pa so zelo nizko točnost načina 5 presegli vsi ostali načini. Pri metodah SMO in Bagging klasifikacijske točnosti načina 5 nismo izboljšali. Rezultate najboljših primerov vsake od petih metod vidimo v tabeli 5.2,



Slika 5.3: Primerjava rezultatov učnih metod in načinov učenja.

Tabela 5.2: Najboljši primeri posameznih učnih metod

Učna metoda	Način učenja	Prioriteta	Klasifikacijska točnost [%]
ANN	4	N_3, N_2, N_4, N_1	86,22
Bagging	5	/	84,02
SMO	5	/	83,74
AdaBoost	1	N_3, N_2, N_1, N_4	83,47
J48	4	N_1, N_3, N_4, N_2	83,47

kjer sta prikazana tudi način učenja in nastavitev prioritete klasifikatorjev na spodnjem nivoju, s katerima je metoda dosegla ta rezultat. Prioritete klasifikatorjev so v tabeli predstavljene kot permutacija oznak, pri kateri prioriteta pada od leve proti desni in kjer posamezne oznake predstavljajo naslednje klasifikatorje:

- N_1 – klasifikator za iskanje poškodbe metalizacije,
- N_2 – klasifikator za iskanje presežka spojne mase,
- N_3 – klasifikator za iskanje primanjkljaja spojne mase,
- N_4 – klasifikator za iskanje neorientiranosti komutatorja.

Tabela 5.3 prikazuje najboljše rezultate posameznih načinov učenja. Prikazana je tudi učna metoda in nastavitev prioritete, s katerima je način učenja dosegel ta rezultat. Vidimo, da se je pri načinih 4 in 5, ki sta najtočnejša, najbolj izkazala metoda ANN. Sledi jima način 1 z metodo AdaBoost, pri načinih 2 in 3 pa je najboljši rezultat dosegla metoda Bagging.

Med vsemi kombinacijami učnih metod, načinov učenja in prioritet klasifikatorjev na spodnjem nivoju se je za najboljšo izkazala metoda ANN pri načinu učenja 4 in prioriteti N_3, N_2, N_4, N_1. Klasifikacijska točnost omenjene kombinacije je dosegla vrednost 86,22% in skoraj za 3% presega najvišjo

Tabela 5.3: Najboljši primeri posameznih načinov učenja

Način učenja	Učna metoda	Prioriteta	Klasifikacijska točnost [%]
4	ANN	N_3, N_2, N_4, N_1	86,22
5	ANN	/	85,39
1	AdaBoost	N_2, N_3, N_1, N_4	83,47
3	Bagging	N_4, N_1, N_3, N_2	81,54
2	Bagging	N_3, N_2, N_1, N_4	79,33

točnost metode J48, ki je predvidena za končno aplikacijo. Čeprav je razlika očitna, zaenkrat še ni smiselno spreminjati delovanja končne aplikacije, saj bo po vgradnji aplikacije v proizvodno linijo zaradi drugačnih pogojev potreben ponoven zajem slik in ponovno učenje, testiranje ter primerjava učnih metod na novi učni množici. Zajem slik bo po vgradnji avtomatiziran in bo omogočal hitro in učinkovito gradnjo velike učne množice, ki bo dala zanesljivejše in točnejše rezultate, na podlagi katerih bo dokončno izbrana uporabljena metoda.

Rezultati in ugotovitve tega diplomskega dela so koristni in bodo služili kot dobra osnova ter vodilo za nadaljnje prilagoditve in izboljšave avtomatskega nadzora kakovosti komutatorjev.

Poglavje 6

Sklep

Glavni cilj diplomskega dela je bil razvoj vgradne aplikacije, ki v realnem času izvaja avtomatski nadzor kakovosti komutatorjev v proizvodnem postopku. Prvi izziv pri tej nalogi je bila paralelizacija pridobivanja atributov iz slik. S pomočjo knjižnice OpenCV in programskega ogrodja OpenCL smo uspešno paralelizirali celotno fazo strojnega vida, ki je časovno najzahtevnejša, in tako omogočili hitro in učinkovito delovanje končne aplikacije.

Druga faza v delovanju aplikacije je klasifikacija posameznega komutatorja na podlagi atributov, pridobljenih iz njegove slike. Paralelizacija te faze ni bila potrebna, saj smo za njeno izvedbo uporabili rešitev, ki za klasifikacijo uporablja odločitvena drevesa in ni časovno prezahtevna. Odločitvena drevesa smo na podlagi učne množice zgradili v programskem okolju Weka in jih kot strukturirano besedilo zapisali v tekstovne datoteke. Te datoteke končna aplikacija razčleni in iz njih izlušči odločitvena drevesa, na podlagi katerih nato klasificira posamezne primere komutatorjev. Tekstovne datoteke lahko spreminjamo in tako ustrezno prilagajamo delovanje odločitvenih dreves tudi po končani fazi razvoja.

Ko smo dosegli pravilno delovanje aplikacije, smo se lotili zbiranja in analize rezultatov. Za namen primerjave smo preizkusili pet načinov učenja v kombinaciji s petimi učnimi metodami, hkrati pa smo opazovali tudi odvisnost klasifikacijske točnosti od prioritet klasifikatorjev za posamezne vrste

napak. Zaradi majhne učne množice smo se preveč optimističnim napovedim izognili s prečnim preverjanjem. Rezultati so pokazali veliko odvisnost klasifikacijske točnosti od uporabljenih načinov učenja in nastavitev prioritet. Ugotovili smo tudi, da odločitvena drevesa, ki so predvidena za končno aplikacijo, niso najtočnejša, zato je v nadaljevanju razvoja smiselno upoštevati druge metode, ki so se izkazale za točnejše.

Aplikacija trenutno deluje na osebnem računalniku, paralelizacija pa je izvedena preko grafične kartice. V nadaljevanju projekta bomo celotno aplikacijo prenesli na platformo STHORM, na kateri bomo izvedli še zadnje popravke, ki bodo zagotovili hitro in zanesljivo delovanje. Sledila bo vgradnja aplikacije v proizvodno linijo. Po vgradnji bosta zaradi spremenjenih pogojev potrebna ponoven zajem slik za učno množico in ponovna gradnja odločitvenega modela, pri kateri bodo ugotovitve tega diplomskega dela v veliko pomoč.

Literatura

[1] (2014) COPCAMS. Cognitive and Perceptive Cameras project. Dostopno na: `http://www.copcams.eu/`

[2] G. U. J. Mottin, M. Cartron, "The STHORM Platform", v knjigi M. Torquati, K. Bertels, S. Karlsson, F. Pacull (ur.), *Smart Multicore Embedded Systems*, New York: Springer, str. 35-43, 2014.

[3] V. Koblar, B. Filipič, "Designing a quality-control procedure for commutator manufacturing", v zborniku *16th International Multiconference Information Society*, Ljubljana, Slovenija, 2013, zv. A, str. 55–58.

[4] (2014) LabVIEW. System design software. Dostopno na: `http://www.ni.com/labview/`

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, "The WEKA Data Mining Software: An Update", *SIGKDD Explorations*, št. 1, zv. 11, str. 10–18, 2009.

[6] (2014) OpenCV. Open source computer vision and machine learning library. Dostopno na: `http://opencv.org/`

[7] (2014) OpenCL. The open standard for parallel programming. Dostopno na: `http://www.khronos.org/opencl/`

[8] W. Burger, M. J. Burge, *Principles of Digital Image Processing: Fundamental Techniques,* London: Springer, 2009.

[9] W. Burger, M. J. Burge, *Principles of Digital Image Processing: Core Algorithms,* London: Springer, 2009.

[10] (2014) Image Formats Compared. Comparison of common image file formats. Dostopno na: `http://www.aivosto.com/vbtips/imageformats.html`

[11] B. Filipič, "Strojno učenje in odkrivanje znanja v podatkovnih bazah (data mining)", *Učno gradivo za predmet Inteligentni sistemi*, Ljubljana: Univerza v Ljubljani, Fakulteta za strojništvo, 2002.

[12] I. Witten, E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques,* 2nd Edition, San Francisco: Morgan Kaufmann, 2005.

[13] J. R. Quinlan, "Discovering rules by induction from large collections of examples", v knjigi D. Michie (ur.), *Expert Systems in the Microelectronic Age*, Edinburgh: Edinburgh University Press, str. 168–201, 1979.

[14] J. R. Quinlan, *C4.5: Programs for Machine Learning,* San Francisco: Morgan Kaufmann, 1993.

[15] (2014) Visual Studio. Integrated development environment. Dostopno na: `http://www.visualstudio.com/`

[16] (2014) Module OCL. OpenCL module within OpenCV library. Dostopno na: `http://docs.opencv.org/modules/ocl/doc/introduction.html`

[17] (2014) Eclipse. Integrated development environment. Dostopno na: `http://www.eclipse.org/`

[18] K. A. Hawick, A. Leist, D. P. Playne, "Parallel Graph Component Labelling with GPUs and CUDA", *Parallel Computing*, št. 12, zv. 36, str. 655–678, 2010.